Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería de Computadores

Trabajo Fin de Grado

Contramedidas para técnicas de detección de Bots en RR.SS.

Autor: Ricardo Stefanescu Abad

Tutor/es: Manuel Sánchez Rubio

2020 - 2021

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

<<Grado en Ingeniería de Computadores>>

Trabajo Fin de Grado
Contramedidas para técnicas de detección de Bots en RRSS.

Autor: Ricardo Stefanescu Abad

Tutor/es: Manuel Sánchez Rubio

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA:

Índice

Ilustraciones

Tablas

Ecuaciones

Anexos

1. Resumen

Los Bots son programas que realizan acciones automáticamente en internet, ejemplo de ellos son los usados para indexar contenido para motores de búsqueda, publicar noticias en Twitter, o con fines menos morales, como propagar desinformación o hacer trampas en juegos online.

Estos últimos han supuesto que se cree una industria basada en desarrollar medidas para detectar estos Bots y detenerlos, de tal forma que no influyan en la experiencia de los usuarios legítimos en estos servicios en internet.

Este proyecto estudia las medidas de detección más prevalentes, y trata de desarrollar un toolkit (caja de herramientas) en Python, que ayuda a programadores con intenciones benignas a crear Bots que son detectados más difícilmente.

Palabras clave:

Redes sociales, Bots, Caja de herramientas, Librería, Paquete, Python

2. Summary

Bots are programs that automatically perform actions on the Internet, examples of which are those used to index content for search engines, publish news on Twitter, or for less moral purposes, such as spreading misinformation or cheating in online games.
The latter has meant that an industry has been created based on developing measures to detect these Bots and stop them, so that they do not influence the experience of legitimate users of these Internet services.
This project studies the most prevalent detection measures and seeks to develop a toolkit in Python that will help programmers with benign intentions to develop Bots that go unnoticed.

Keywords
Social networks, Bots, Toolkit, Library, Package, Python

3. Resumen extendido

El progreso de la civilización humana siempre ha ido ligado a su capacidad de innovar, y transformar problemas complejos en triviales.
La revolución de internet en nuestras vidas ha supuesto que podamos controlar aspectos financieros, informativos y sociales desde cualquier lugar si tenemos acceso a él. Como todo lo relacionado con ordenadores, estas acciones pueden ser automatizadas.

Los Bots son programas que realizan acciones automáticamente en internet, y simplifican muchas tareas, como las de pedir productos por internet con Amazon Dash, o recopilar información para estudios estadísticos. También supone que puedan ser usados con fines malignos, como el spam, o la propagación de desinformación.

El uso de Bots con fines malignos ha supuesto la creación de una industria basada en detectarlos y mitigar sus acciones. La vasta cantidad de medidas creadas para detectarlos ha supuesto una carrera armamentística en torno a crear medidas y contramedidas.

En este trabajo pretendemos asistir a los programadores con intenciones benignas, para ello crearemos una librería de Python que asiste a la programación de Bots creando funcionalidades para que se mimeticen con las cuentas legitimas.

4. Glosario de acrónimos y abreviaturas

RRSS	REDES SOCIALES Medio social que permite establecer contacto con otras personas por medio de una plataforma web
LN	LENGUAJE NATURAL Aquellas lenguas generadas espontáneamente en un grupo de hablantes con propósito de comunicarse.
MM	MOUSE MOVEMENT Movimiento de cursor sin clics.
PC	POINT AND CLICK Movimiento de cursor que finaliza con un clic.
DD	DRAG AND DROP Movimiento de cursor que comienza presionando el botón, y finaliza soltándolo.
GAN / ANN	GENERATIVE ADVERSARIAL NETWORK Modelo de red neuronal basado en varias redes que compiten entre sí.

5. Introducción

Llamamos Bots sociales a aquellos que controlan cuentas en RRSS, los primeros ejemplos de se remontan a los ChatBots, que contestaban mensajes automáticamente en servicios como Messenger.
Actualmente las razones para diseñarlos son variadas, y no todas buenas. Un artículo de Forbes [1] identifica 5 usos malvados para los Bots sociales:

- Simular fama teniendo muchos seguidores controlados por Bots sociales.
- Publicar publicidad encubierta.
- Acosar a personas.
- Influenciar la opinión pública.
- Limitar el libre discurso.

Estos usos han supuesto que las RRSS busquen medidas para detectar y mitigar las acciones de cuentas controladas por Bots. [2]

Algunas RRSS permiten el uso de Bots mientras que no tengan finalidades malignas, como Twitter, mientras que otras prohíben completamente las cuentas controladas por Bots.

5.1 Objetivos

Nosotros creemos que no todos los Bots sociales son creados con fines malignos.
Este trabajo estudia las medidas usadas por los servicios web para detectar Bots, y propone una serie de contramedidas con las que seamos capaces de programar Bots que se camuflan como usuarios legítimos.

La librería estará en un paquete de funciones en Python, organizado de manera modular para que podamos seguir añadiendo funcionalidades en un futuro.
Tenemos como objetivo implementar una primera versión de las siguientes funcionalidades:

- Sintetizar movimientos de mouse de apariencia natural.
- Sintetizar tecleo con cadencia natural.

- Imitar ritmos temporales humanos.
- Reaccionar a contenido dados unos intereses.
- Sintetizar contenido que simula el humano.

5.2 Metodología

Por cada medida:

1. Estudiaremos cómo funcionan las medidas de detección.
2. Propondremos cómo funcionarán nuestras contramedidas.
3. Las desarrollaremos en cuadernos interactivos de Python.
4. Implementaremos las funciones en una librería en Python.

Para investigar las medidas recopilaremos información de artículos científicos, publicaciones de compañías dedicadas a la detección de Bots, y otras contramedidas ya existentes.

6. Medidas de detección

6.1 Clasificación de medidas

Podemos dividir las medidas de detección en dos grupos, aquellas que pueden ser automatizadas, y las que requieren humanos. Un ejemplo de la primera son aquellas medidas que aplican aprendizaje máquina para calcular la probabilidad de que una cuenta sea un Bot; y un ejemplo de la segunda son los moderadores, y la capacidad de los usuarios de reportar cuentas. En este trabajo trataremos ambas.

Las medidas de detección automáticas necesitan ser fiables antes de tomar una decisión que pueda afectar a un usuario real. Las pruebas más fiables son las pruebas de Turing automáticas (Captchas), que presentan a la cuenta sospechosa con una actividad simple para un humano, pero muy compleja para un Bot.

Ilustración 1: Captcha de LinkedIn

Estas medidas son considerados intrusivos, ya que suponen una interrupción del uso habitual de la aplicación, y pueden estropear la experiencia de los usuarios legítimos.
Antes de usar una medida intrusiva, las RRSS usan medidas no intrusivas, recolectando y procesando datos pasivamente, como el comportamiento de los usuarios, y las comunidades que frecuentan.
En este trabajo estudiaremos las medidas no intrusivas.

Las medidas no intrusivas estudian el comportamiento de los usuarios; está compuesto por infinidad de factores, como los horarios de uso, y la cantidad de contenido generado. También se estudian las interacciones con la aplicación, como los movimientos de ratón, la cadencia de escritura y las pausas.

Otra medida no intrusiva es estudiar el comportamiento del usuario fuera de la aplicación. Los servicios web son capaces de rastrear que hacen los usuarios en internet gracias a su huella digital, que calculan a partir de los rasgos identificativos de su dispositivo.

Como decíamos anteriormente, no todos son sistemas automáticos. La mayoría de las RRSS incorporan sistemas por los cuales usuarios pueden reportar el comportamiento sospechoso de otros; si una cuenta es reportada el suficiente número de veces, pasa a ser estudiada por un moderador humano que determina si la cuenta está siendo controlada por un Bot.

6.2 Contramedidas

"Hecha la ley, hecha la trampa", los Bots sociales modernos incorporan toda clase de métodos para evitar ser detectados, como formas de imitar el comportamiento humano y formas de resolver pruebas intrusivas. Llamaremos a estos métodos contramedidas.

En este trabajo implementaremos las siguientes contramedidas en un paquete:

- Síntesis de trayectorias de mouse realistas.
- Síntesis de cadencia de tecleo realista.
- Síntesis de periodos de activación humanos.
- Cálculo de la reacción a un texto dados unos intereses.
- Retoque de fotos automático para generar fotos realistas.
- Síntesis de lenguaje natural dados unos temas.

Además, estudiaremos las contramedidas realizables en una futura versión de este paquete.

7. Biometría del comportamiento

7.1 Introducción

Llamamos biometría del comportamiento al estudio de los patrones medibles e identificativos en las actividades humanas.

En las guerras mundiales, era común el uso del telégrafo como herramienta de comunicación. A pesar de no tener garantías de quien estaba al otro extremo de la línea de telégrafo, los telegrafistas eran capaces de reconocer quien era el emisor del mensaje solo por la cadencia con la cual tecleaba los puntos y líneas. A este estilo de teclear se le llamaba el "puño" del emisor, y es uno de los primeros ejemplos de biometría del comportamiento. Se aplico para garantizar la seguridad de las líneas de telégrafo, así como para rastrear movimientos de barcos y submarinos.

En la actualidad no solemos usar telégrafos, pero sí ratones y teclados; tal y como pasaba con el telégrafo, la forma con la que tecleamos o movemos el ratón puede ser usada para identificar usuarios individuales. Productos como castle.io [3] usan la biometría del comportamiento para detectar Bots.

En este trabajo vamos a estudiar los patrones identificativos del uso del ratón y el teclado.

7.2 Biometría de movimientos del Ratón

7.2.1 Introducción

El 9 de diciembre de 1968, D. Engelbart presentó lo ahora conocido como "La Madre de Todas las Demos" ["The Mother of All Demos"] donde demostró múltiples tecnologías entonces experimentales, como el hipertexto, las videoconferencias y los editores de texto colaborativos.
Entre estas tecnologías entonces consideradas futuristas, se encontraba una especialmente relevante para nuestro trabajo, el primer prototipo de un ratón para una computadora.

Al igual que otras tecnologías presentadas en "La Madre de Todas las Demos", el ratón fue adoptado como un estándar en todo ordenador personal. Actualmente existen otras formas de mover un cursor por pantalla, como los paneles táctiles, que terminan siendo lo mismo a efectos prácticos.

Los movimientos del cursor por la pantalla pueden ser estudiados para tratar de discernir entre aquellos que son sintéticos y aquellos producidos por un uso natural de un ratón [3]. Hay diferentes propuestas sobre realizar esta distinción, se pueden usar pruebas estadísticas como Kolmogórov-Smirnov, SVM o redes neuronales [4]

Las trayectorias del cursor por la pantalla son altamente complejas, y no pueden ser expresadas con una simple expresión matemática. Por ello es necesario que las separemos en características [features] con valores atómicos que podamos estudiar.

Cada artículo publicado propone diferentes características para clasificar los movimientos, a continuación, agrupamos las más comunes. [5] [6]

Nombre	Unidad
Tipo de movimiento	MM - PC - DD
Velocidad	Pixeles/Segundo
Aceleración	Pixeles/Segundo^2
Angulo inicial	Radianes
Angulo final	Radianes
Varianza de velocidad	Pixeles/Segundo
Varianza de aceleración	Pixeles/Segundo ^2
Desviación de un movimiento rectilíneo	Pixeles
Tiempo	Segundos

Tabla 1: Características de las trayectorias del cursor

7.2.2 Planteamiento de la solución

Ahora que conocemos las características estudiadas, decidiremos cómo vamos a estructurar un sistema que genera trayectorias que pasen desapercibidas por humanas.

Existen varias librerías de código abierto que generan trayectorias con características humanas, entre ellas se encuentran Natural Mouse Motion [7], o BezMouse [8]. Hay trabajos que usan sistemas complejos con múltiples fases [9]. Y sistemas que usan redes neuronales para imitar trayectorias humanas como demuestran autores como *'neuronio'* [10].

Nuestro primer desafío es generar movimiento no rectilíneo, ya que ningún movimiento de mouse realizado por una persona lo es. La librería Natural Mouse Motion genera trayectorias con curvas sinusoidales, BezMouse usa curvas de Bézier y los otros sistemas aplican redes neuronales. Estos últimos son los que mejor pasan desapercibidos por las medidas de detección, pero también los más complejos de implementar.

El segundo desafío es generar imperfecciones en el movimiento, como cuando movemos el ratón por una superficie con relieve que no detecta bien, o pequeños espasmos musculares. Natural Mouse Motion implementa una función que añade ruido y desviamientos, y BezMouse realiza variaciones de velocidad.

Para nuestra solución hemos decidido plantear un sistema que consiste en tres fases, la primera genera una trayectoria mediante curvas de Bézier, la segunda imita la aceleración inicial y desaceleración final tomando puntos aleatorios de una distribución triangular, y la tercera inyecta ruido para romper la continuidad.

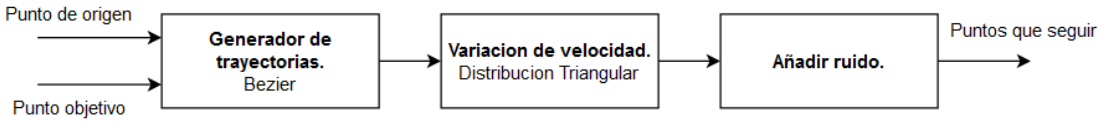

Ilustración 2: Esquema del sistema para movimientos del cursor

7.2.3 Implementación del sistema

El código estará implementado en nuestro paquete ***natural_Bot_tk*** dentro del submódulo ***natural_input.***

```
.
└── natural_bot_tk
    ├── natural_input
    │   ├── **Mouse**
    │   └── Keyboard
    ├── scheduling
    ├── reaction
    ├── text_generation
    └── img_generation
```

Ilustración 3: Localización del sistema de Mouse

Para que sea más intuitivo, mostraremos cómo varía la trayectoria entre dos puntos según evoluciona nuestro sistema. Como base, presentamos una trayectoria rectilínea.

Ilustración 4: Trayectoria rectilínea

Las curvas de Bézier son curvas paramétricas determinadas por uno o más puntos de control, se usan ampliamente en los gráficos por ordenador para representar curvas a partir de sus puntos de control.

La fórmula de las curvas de Bézier nos permite calcular puntos discretos de la curva dado un valor ***t*** comprendido en el intervalo ***(0, 1)***.

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , 0 \leq t \leq 1.$$

Ecuación 1: Curva de Bézier cubica

Donde:

- $\mathbf{P_0}$ es el punto en el origen
- $\mathbf{P_1}$ y $\mathbf{P_2}$ son los puntos de control
- $\mathbf{P_3}$ es el punto final.

Programamos una función que calcula N puntos de una curva de Bézier con puntos de control aleatorios:

```
def generate_cubic_bezier(self, p_0, p_3,
                                max_deviation=0.05,
                                n_steps=None,
                                linear_progression=False):
        ''' Calcula los puntos de una curva de Bézier.\n
        Los parámetros son:\n
        `p_0`: El punto inicial e.g. [0,0]\n
        `p_3`: El punto final e.g. [10,10]\n
        `max_deviation`: La desviación máxima de los puntos de
         control\n
        `n_steps`: El número de puntos de la curva que calcular\n
        `linear_progression`: \n
          - True: usamos una distribución lineal para los puntos\n
          - False: Usamos una distribución triangular para los
                   puntos
        '''
```

Esta función genera dos puntos de control aleatorios y retorna ***n_steps*** puntos de la curva. Podemos ver un ejemplo de los puntos resultantes al llamar la función.

generate_cubic_bezier(p_0, p_3, max_deviation=0.5, n_steps=60, linear_progression=True)

Ilustración 5: Puntos de una curva generados por nuestra función (lineal)

Estos puntos representan por donde pasará nuestro mouse, para simular aceleración y desaceleración variamos la distribución de puntos en la curva. Donde queramos que el ratón se mueva más lento, calcularemos más puntos, y viceversa.

Para ello extraemos los valores ***t*** de una distribución aleatoria triangular, con el pico cerca del final. Una distribución aleatoria triangular tiene más probabilidad de generar valores cerca de su pico, en nuestro caso, simulando el ralentizamiento del ratón.

$$f(x|a,b,c) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & \text{para } a \leq x < c, \\ \frac{2}{b-a} & \text{para } x = c, \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{para } c < x \leq b, \\ 0 & \text{para otros casos} \end{cases}$$

Ecuación 2: Distribución triangular de probabilidad

Para nuestra finalidad, usaremos la implementación de la librería ***numpy*** ***numpy.random.triangular***.

A continuación, mostramos un ejemplo de la misma curva donde hemos extraído los puntos de una distribución triangular.

generate_cubic_bezier(p_0, p_3, max_deviation=0.5, n_steps=60, linear_progression=False)

Ilustración 6: Puntos representando cambio de velocidad

Por último, queremos simular pequeños espasmos musculares, o superficies imperfectas. Para ello necesitamos introducir ruido en la curva.

Hemos programado una función que introduce ruido:

```
def add_noise(self, points, noisiness, max_deviation):
    '''
    `points` - Puntos a los que añadir ruido \n
    `noisiness` - Valor [0-1) que determina a qué porcentaje de
puntos
     añadir ruido \n
    `max_deviation` - Desviación máxima
    '''
```

El desafío de introducir ruido es que el ruido total debe sumar 0; si no, la trayectoria no alcanzara nuestro objetivo.
Para ello seleccionamos pares de puntos aleatorios, y generamos pequeñas desviaciones aleatorias en forma de vectores. Por último, añadimos estas desviaciones a los pares de puntos, pero con direcciones opuestas, consiguiendo que el ruido se cancele a pares.

A continuación, podemos ver que sucede aplicando ruido sobre los puntos anteriores.

add_noise(points, 0.9, 0.6)

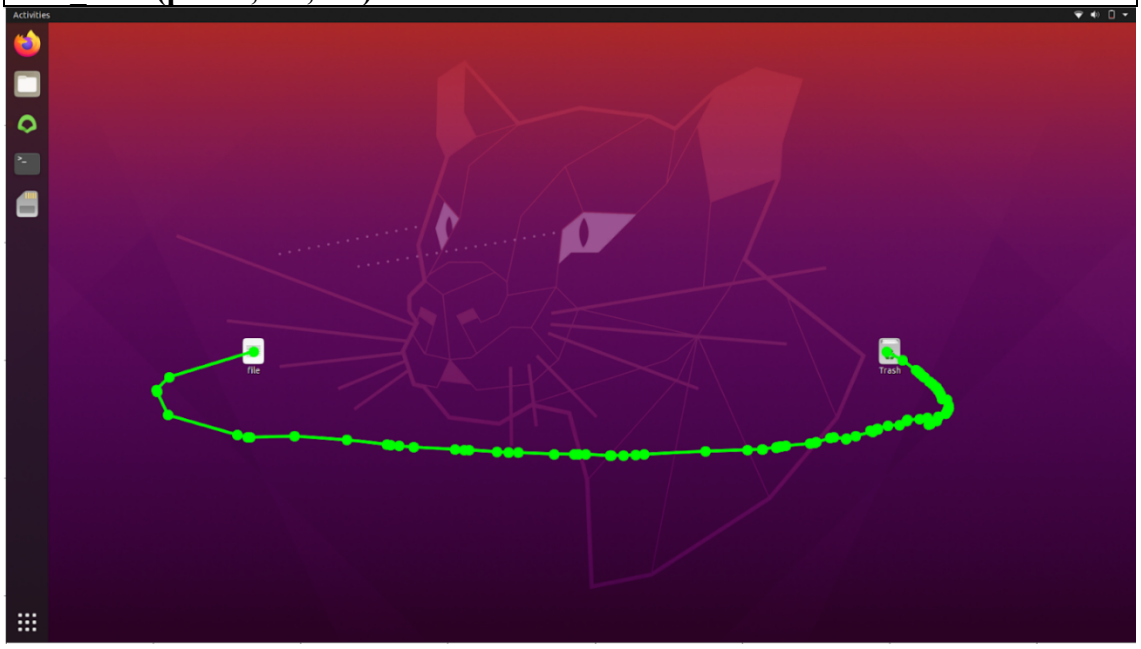

Ilustración 7: Trayectoria final

7.2.4 Resultados y futuras mejoras

A continuación, podemos ver algunos ejemplos de trayectorias creadas con nuestro sistema.

Las medidas actuales emplean sistemas de aprendizaje continuo con un flujo constante de datos de humanos reales, por lo que cada vez será más difícil engañarlas; nuestra solución presenta una contramedida efectiva contra medidas simples, el sistema por fases supone que podremos mejorar el conjunto mejorando fases individuales.

7.3 Biometría del tecleo

7.3.1 Introducción

Como hablamos en la introducción, la cadencia del teclado puede ser usada para discernir entre diferentes usuarios y Bots. Se aplica como medida para añadir seguridad extra al autenticar usuarios [11], detectar intrusiones en sistemas [12], y como en nuestro caso, detectar Bots.

Cada persona tiene su propia forma de teclear, las personas mayores escriben con un dedo buscando las teclas, mientras que las personas más jóvenes usan diferentes dedos para diferentes teclas. Estos cambios suponen sutiles variaciones en el tiempo que se tarda de tecla a tecla, el número de palabras por minuto, y otras muchas características que pueden ser usadas para discernir entre diferentes usuarios y Bots.

Para clasificar usuarios según estas características, se suelen usar algoritmos de aprendizaje no supervisado.

Igual que hicimos con los movimientos de mouse, listamos las características más comúnmente usadas para crear clasificadores en la literatura. [13]

Nombre	**Unidad**
Tiempo de vuelo entre dos teclas	milisegundos
Tiempo que presionas una tecla	milisegundos
Duración total para escribir X palabra	milisegundos
Palabras por minuto	WPM

Tabla 2: Características de la cadencia de tecleo

Los sistemas de clasificación funcionan mejor cuando agrupan estas características en grupos de 3 caracteres, llamados trigrafos. Por ejemplo, estudiar el tiempo de vuelo entre las teclas ***a - b - c*** en vez de solo ***a - b***. [14]

Detectar Bots no es diferente a discernir entre usuarios. Mientras que cada usuario suele tener su propia cadencia al teclear, Bots con sistemas de tecleo simples pulsan teclas a un ritmo demasiado estable o demasiado aleatorio, lo que es fácilmente detectable por un sistema que emplea métodos estadísticos.

7.3.2 Planteamiento de la solución

A la hora de simular el presionado de teclas con cadencia humana, no existen muchas implementaciones públicas. Una de las mejores herramientas para crear Bots sociales, *Browser Automation Studio*, que simula movimientos de ratón humanos, no incorpora ninguna funcionalidad para simular tecleo humano. [15]

Nuestro objetivo es generar un sistema por el cual un Bot siga una cadencia predeterminada con pequeñas variaciones al teclear texto.
Al igual que con el sistema para generar movimientos de mouse, será un sistema con múltiples fases independientes.

El sistema se dividirá en tres fases, las dos primeras generan la cadencia personalizada del Bot, y la última generara la secuencia de teclas a ser presionadas.

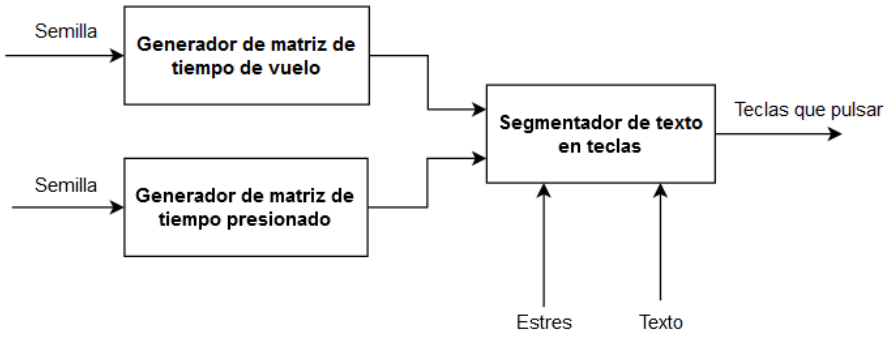

Ilustración 8: Esquema del sistema de tecleo

7.3.3 Implementación del sistema

El código estará implementado en nuestro paquete ***natural_Bot_tk*** dentro del submódulo ***natural_input.***

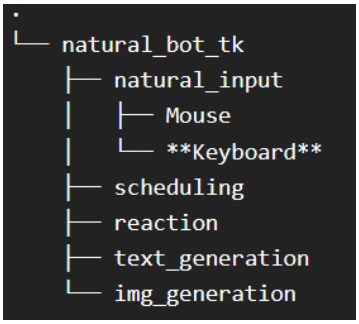

Ilustración 9: Localización del sistema de tecleo

Para generar una cadencia personalizada necesitamos, los tiempos de vuelo normales entre cualquier combinación de dos teclas, y el tiempo que se mantiene presionada cada tecla.

Para calcular los tiempos de vuelo hemos programado la siguiente función:

```
def _generate_move_delay_matrix(self):
    ''' Calcula una matriz de tiempo de vuelo aleatoria '''
```

Esta función nos devuelve una matriz de tiempos entre dos teclas de un teclado simulado. La calcula a partir de las coordenadas de las teclas, de las cuales sacamos la distancia euclídea y le sumamos un pequeño valor aleatorio.

El cálculo de cuánto tiempo queda cada tecla presionada, es completamente aleatorio, y es realizado por la siguiente función:

```
def _generate_hold_delay_dict(self):
    ''' Calcula un diccionario de hold delay aleatorio '''
```

Una vez tenemos estos tiempos calculados, queremos ser capaces de calcular la secuencia de caracteres que serán pulsados para escribir un texto, con un extra, añadiremos también la capacidad de cometer errores y corregirlos.

Calculamos la secuencia de caracteres y tiempo se calcula con la siguiente función:

```
def generate_keys(self, text, stress):
''' Genera la secuencia de teclas y sus tiempos
 para teclear un texto. \n
`text`: El texto a teclear\n
`stress`: Valor que determina la velocidad y la cantidad de
  errores.'''
```

Un ejemplo, queremos teclear el texto “Hola caracola”, el resultado de llamar a esta función con un valor de stress de 0.2 genera:

generate_keys(“Hola caracola”, 0.2)

	Tecla	Mayus	Tiempo entre teclas	Tiempo presionado
0	h	1	0	0.1764
1	o	0	0.0819	0.1557
2	l	0	0.0536	0.1003
3	a	0	0.2501	0.0704

4	space	0	0.1498	0.1610
5	c	0	0.0867	0.1306
6	a	0	0.0845	0.0704
7	r	0	0.0826	0.1514
8	a	0	0.0826	0.0704
9	c	0	0.0826	0.1306
10	o	0	0.1790	0.1557
11	l	0	0.0536	0.1003
12	a	0	0.2501	0.0704

Tabla 3: Secuencia de teclas con valor de estrés = 0.2

Mientras que llamarla con un valor stress de 0.5 genera lo siguiente:

generate_keys(“Hola caracola”, 0.5)				
	Tecla	**Mayus**	**Tiempo entre teclas**	**Tiempo presionado**
0	h	1	0	0.1764
1	o	0	0.0819	0.1557
2	l	0	0.0536	0.1003
3	a	0	0.2501	0.0704
4	space	0	0.1498	0.1610
5	c	0	0.0867	0.1306
6	a	0	0.0845	0.0704
7	r	0	0.0826	0.1514
8	a	0	0.0826	0.0704
9	a	0	0.0826	0.0704

10	backspace	0	0.3807	0.1793
11	backspace	0	0.0472	0.1793
12	a	0	0.0826	0.0704
13	c	0	0.0826	0.1306
14	o	0	0.1790	0.1557
15	l	0	0.0536	0.1003
16	a	0	0.2501	0.0704

Tabla 4: Secuencia de teclas con valor de estrés = 0.5

Como podemos ver en las casillas marcadas, cuando teclea dos veces la segunda ‘a’ de caracola, decide borrarlas y continuar.

7.4 Introduciendo Input al sistema

Las clases **Mouse** y **Keyboard** implementan funciones para mover el cursor por los puntos, o pulsar las teclas a ritmo usando el paquete ***PyAutoGUI*** [16].

Las funciones en concreto son:

```
    def move_to(self, p_destination,
                max_deviation=0.05,
                n_steps=None,
                linear_progression=False,
                noisiness=0.4,
                max_noise_deviation=0.9):
        '''
        Mueve el ratón a un píxel dadas sus coordenadas.\n
        `max_deviation`: La desviación máxima de los puntos de
         control\n
        `n_steps`: El número de puntos de la curva que calcular\n
        `linear_progression`: \n
          - True: usamos una distribución lineal para los puntos\n
          - False: Usamos una distribución triangular para los
                   puntos\n
        `noisiness` - Valor [0-1) que determina a qué porcentaje de
                      puntos
         añadir ruido \n
        `max_noise_deviation` - Desviación máxima
        '''
```

Y

```
    def type_text(self, text, stress=0.3):
        ''' Genera las teclas para el texto y las teclea\n
        `text`: El texto a teclear\n
        `stress`: Determina la cantidad de errores y la velocidad'''
```

Se puede ver su funcionamiento interno en el Anexo X para la clase **Mouse** y en el Anexo Y para la clase **Keyboard**.

Se pueden ver ejemplos de su uso en los notebooks en el Anexo X para la clase **Mouse** y en el Anexo Y para la clase **Keyboard**.

8. Coherencia temporal

8.1 Ritmos diarios

Los ritmos circadianos, del latín *circa* (“alrededor de”) y *dies* ("día”), son los ciclos sueño-vigilia de 24 horas por los que pasamos los humanos.
Además de este ciclo de sueño, la mayoría de los humanos tenemos una ocupación que ronda las 8 horas, nos gusta tomarnos un café por las mañanas, y tenemos la mayoría de nuestro tiempo libre por la tarde noche.

Si estos patrones temporales se ven reflejados en nuestro uso de RRSS, ¿por qué no en el de nuestros Bots?

8.2 Aplicación para la detección de Bots

Una forma de detectar Bots dedicados a hacer spam contenido, es observar sus patrones temporales. Es fácil para sistemas automáticos y moderadores detectar las cuentas que publican contenido sin descanso, o aquellas que lo publican todos los días a las 5 de la tarde religiosamente.

8.3 Solución propuesta

Nuestro objetivo es generar un sistema por el cual seamos capaces de calcular tiempos de activación coherentes para un Bot.
Para ello hemos propuesto un sistema que genera distribuciones de probabilidad con las cuales los programadores generar de forma procedural tiempos de activación para sus Bots.

8.4 Implementación

El código estará implementado en nuestro paquete **natural_Bot_tk** dentro del submódulo **natural_input.**

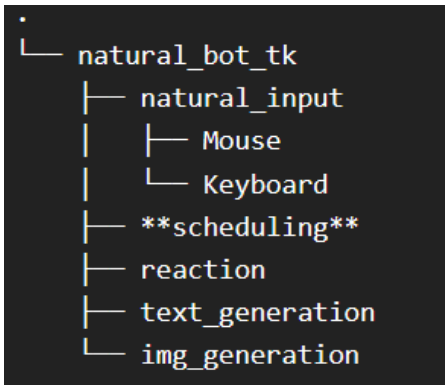

Ilustración 8: Localización del sistema de tiempos de activacion

Para implementar este sistema nos hemos basado en cómo se planifican procesos con diferentes prioridades en un procesador. En vez de usar procesos, usaremos *Ocupaciones.*

Una Ocupación es un objeto que contiene:

- Su prioridad.
- El número de unidades de tiempo que tarda (minutos).
- Probabilidad de que se active el Bot.
- (Opcional) Cuando empieza.

La probabilidad de una ocupación representa la probabilidad de la densidad de probabilidad mientras la ocupación esta activa.

El sistema toma una semilla para un generador de números pseudoaleatorios, que usara para generar una serie de ocupaciones. Estas ocupaciones serán:

- Una tarea de baja probabilidad (simulando trabajo) de 5 a 8 horas.
- Una tarea de muy baja probabilidad (simulando sueño) de 6 a 12 horas.
- Varias tareas de media probabilidad (simulando quehaceres) de 1 hora al tiempo restante (en total).
- Varias tareas de muy alta probabilidad (simulando tiempo libre) de 1 hora al tiempo restante.

Por último, usa un algoritmo de programación preventiva de prioridad fija para generar una probabilidad por cada unidad de tiempo, en nuestro caso minutos, y lo encapsula en una función que podemos usar para generar números aleatorios dentro de nuestra distribución.

Este procedimiento está dentro de la siguiente función:

```
def generate_scheduling_function(person_seed,
                                 max_day_var=0.1,
                                 sleep_activation_prob = 0.02,
                                 high_occupation_prob = 0.1,
                                 mid_occupation_prob = 0.3,
                                 low_occupation_prob = 0.6):
   ''' Retorna una función con el prototipo
   day_func(day_seed, amount=1)
   para generar en qué minuto/s se debe activar el Bot.
   `person_seed`: Seed del Bot para generar su horario \n
   `max_day_var`: Máxima variación por día \n
   `sleep_activation_prob`: Probabilidad de que se active durmiendo
                             \n
   `high_occupation_prob`: Probabilidad de que se active estando
                            muy ocupado \n
   `mid_occupation_prob`: Probabilidad de que se active estando
                          medianamente ocupado \n
   `low_occupation_prob`: Probabilidad de que se active estando
                          poco ocupado'''
```

generate_scheduling_function retorna una función que nos dará los minutos en los cuales tenemos que activar nuestro Bot. Esta función retornada tiene el prototipo

day_func(day_seed, amount=1), donde ***day_seed*** es una semilla para el generador de números pseudoaleatorios, representando el día; y ***amount*** el número de activaciones que queremos calcular.

Un ejemplo de la distribución de la densidad de un día generada por este sistema es el siguiente:

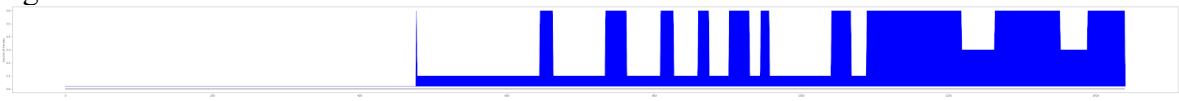

Ilustración 9: Probabilidades de activación a lo largo de un día.

El eje X es el momento del día en minutos, y el eje Y es la probabilidad de generar un valor.

Por último, convertimos estos valores en una distribución de probabilidad, y extraemos minutos aleatorios para que se active el nuestro Bot.

Podemos ver el código en un notebook en el anexo X, y un ejemplo de uso en el anexo Y.

9. Huella digital

9.1 Que es

Hasta ahora hemos implementado tres sistemas que intentan esconder las acciones de Bots por cómo interactúan con el sistema. Presentaban soluciones para medidas de detección visibles; a continuación, trataremos sobre la huella digital, una serie de rasgos identificativos invisibles de los dispositivos, que pueden ser usados para desenmascarar Bots.

Los Bots sociales tienen como objetivo aplicaciones web, que están basadas en el protocolo HTTP y el lenguaje HTML.
HTTP y HTML nacieron en los 90 con la necesidad de crear estándares para comunicar computadoras por internet. Para estos estándares, aparecieron multitud de navegadores web que los soportaban; pero su rápida evolución supuso que no todos los navegadores fueran capaces de soportar todas las instrucciones de los estándares, algunos incluso añadiendo sus propias funcionalidades.
Para evitar problemas de compatibilidad, el protocolo HTTP incluyó un atributo con el nombre y versión del navegador. Este, aunque ahora arcaico, es la primera base de una huella digital, ya que presenta a la página web con datos sobre nuestro dispositivo.
Estos pequeños trozos de información sobre nuestro dispositivo pueden ser usados para identificarnos a través de diferentes páginas web.

Además de toda la información que nuestro navegador envía, como cabeceras HTTP o cookies; existen las características identificativas de cada dispositivo, estas pueden ser extraídas con pequeños scripts incrustados en las páginas web que visitamos.

Las páginas web actuales suelen implementar código de terceros como Google o Facebook. Este código, implementado por razones ajenas a nuestro trabajo, permite a (por ejemplo) Google calcular una huella digital del dispositivo del usuario, y uniendo los pedacitos de información que recopilan de las páginas que usan el código de Google, reconstruir sus hábitos en internet.

Otro uso de la huella digital es, como el lector se podrá imaginar, la detección de Bots.

9.2 Su uso para detectar Bots

La huella digital del dispositivo pasa muchas veces desapercibida a la hora de diseñar Bots, y sin embargo puede ser uno de los mayores factores a la hora de detectarlos.
Mientras que un usuario normal, tiene una huella digital peculiar, los Bots programados con librerías como ***Selenium*** suelen tener una muy similar entre ellos, ya que usan un navegador con apenas extensiones y cambios de configuración.

Una opción es crear Bots en máquinas virtuales que se hacen pasar por otros dispositivos, como IPhone o navegadores diferentes. Actualmente existen productos dedicados a detectar estos intentos mediante la huella digital, como Google Picasso
[17].

Otras medidas de detección estudian a los usuarios continuamente a través de distintas páginas web para comprobar si un usuario es un Bot, y ser capaces de bloquearlos de distintas funcionalidades. Uno de los factores que *Google ReCaptcha V3*, es si el usuario había interactuado con servicios de Google antes, y como actuaba [18].

Por último, hay productos para detectar Bots [19] que comparten huellas digitales de Bots como forma de inteligencia, permitiendo una más fácil detección a través de diferentes servicios.

9.3 Que compone una huella digital

No existe ningún estándar sobre los atributos que componen una huella digital, ya que difieren según la implementación del sistema que la calcula.
A continuación, listamos atributos de la huella digital que nos interesa tener en mente a la hora de programar Bots.

9.3.1 Cabeceras HTTP

Como comentamos en el ejemplo anterior, nuestro navegador envía una serie de campos en las peticiones HTTP, estas contienen pequeños datos de nuestro dispositivo como, por ejemplo:

- **User agent**
- **Accept**
- **Content encoding**
- **Content language**

Estas pueden ser fácilmente falseadas, solo hay que cambiar el contenido que envía nuestro navegador al realizar una petición.

A la hora de desarrollar Bots nos interesa que sean valores coherentes (Si usamos un generador de trayectorias de ratón, que no digan que fue un dispositivo móvil), y que se mantengan constantes o con muy ligeras variaciones a lo largo del uso del Bot.

9.3.2 Cookies

Las cookies, al igual que las cabeceras, son datos que se envían en las peticiones HTTP. Estos datos, como tokens de sesión, son creados por las páginas web, y guardados de manera local por nuestro navegador.

Al contrario de las cabeceras, las cookies solo se envían a dominios seleccionados.
Si una página usa recursos de otras webs, como anuncios, puede guardar y recibir cookies de dominios diferentes, facilitando el tracking.

Un usuario que usa el internet de forma natural suele tener multitud de cookies de distintos servicios, así como algunas de tracking. Para simular esto debemos conseguir que nuestros Bots usen distintos servicios de internet.

9.3.3 Extensiones de navegador

Mientras que nuestro navegador no envía ninguna clase de información sobre las extensiones instaladas a las páginas que visitamos, es fácil detectarlas si la página web tiene ese cometido.
Las extensiones guardan sus recursos con una URL estándar como la siguiente **"extension://<extensionID>/<pathToFile>"**. Algunas extensiones permiten a sus recursos ser accedidos desde una página web, por tanto, es trivial detectar su presencia.
Otras no, por tanto, se usan ataques donde se calcula la diferencia del tiempo de respuesta al acceder a un recurso que no existe, y un recurso al cual no tenemos acceso. Con este método se puede detectar el uso de casi cualquier extensión [20] .

9.3.4 Diferencias de renderización

Hay recursos que deben ser renderizados antes de ser mostrados por pantalla, como por ejemplo Canvas de HTML5, diseños 3D con WebGL, o fuentes. El resultado final es ligeramente diferente según el motor de renderizado y el hardware del dispositivo.

Ilustración 10: Ejemplo HTML5 Canvas

Esta ligera variación es suficiente para poder calcular huellas digitales identificativas, y puede ser explotado por sistemas como Google Picasso para discernir entre lo que nuestro Bot dice ser (HTTP Headers) y lo que realmente es (Hardware/Máquina virtual).

9.4 Evolución temporal

La huella digital de un usuario no es estática, un estudio realizado por FP-Stalker detalla que hay tres tipos de variaciones de la huella digital:

- Cambios por actualizaciones de software.
- Cambios por cambios en la configuración del dispositivo.
- Cambios por cambio del entorno del usuario.

Si nuestro objetivo es programar Bots que se hacen pasar por humanos, debemos procurar que la huella digital varíe de forma natural. Para ello, al programar nuestro Bot podríamos:

- Hacer que cambie ligeramente las configuraciones del navegador.
- Que visite otras páginas a parte del objetivo para cambiar sus cookies.
- Que instale y desinstale extensiones del navegador.
- Que actualice el software de la máquina.

9.5 Conclusión

En el anteproyecto planteamos la posibilidad de crear una estructura de datos para guardar una huella digital. Después de esta investigación nos hemos dado cuenta de que no serviría ningún propósito, ya que depende de la implementación del Bot en cuestión, por lo que actualmente no hay nada en lo que podamos asistir desde un paquete de Python.

Aun así, es un campo importante, y se plantea para futuras versiones del paquete.

10. Reactividad a contenido

10.1 Introducción

El uso normal de las RRSS incluye interactuar con el contenido publicado por otras cuentas, si te gusta, reaccionas a él dándole un "*like*" o comentando, y si la red social es capaz de darte más contenido que te gusta, pasas más tiempo en ella.

Por ejemplo, si eres una cuenta que se dedica a publicar contenido sobre impresión 3D, lo normal es que interacciones con otras cuentas que publican contenido similar. Al diseñar Bots queremos emular el mismo comportamiento.

Cómo interactuamos con otras cuentas depende de la implementación. En nuestro paquete vamos a simplificar ese trabajo generando un sistema para reaccionar a contenido. Pero primero explicaremos cómo se detectan los Bots en RRSS por sus comunidades, y que supone reaccionar a contenido.

10.2 Detección de Bots por grupos sociales

La premisa de las RRSS es conectar con cuentas con intereses similares a los tuyos, cuantos más seguidores tengas, a más personas le interesa tu contenido, y más te recomendará la red social.

Una forma fácil de detectar Bots es encontrar cuentas muy activas sin seguidores, o cuentas cuyos seguidores son otras cuentas muy similares, formando una red hermética al exterior.
Por ello una cosa importante si queremos crear Bots que pasen desapercibidos en RRSS, es que estos sean capaces de relacionarse con cuentas reales, y que sigan un patrón de gustos.

10.3 Que es la reactividad al contenido

Llamamos reactividad al contenido a la capacidad de un Bot social de generar una reacción dado el contenido que una red social le presenta.

Por ejemplo, un Bot de Twitter con el cometido de comentar una medallita en las publicaciones que hacen comentarios positivos sobre la película "*Space Jam*". Este Bot reaccionaria favorablemente a los tweets positivos sobre dicha película, mientras que no reaccionaria a los comentarios negativos.

Este sistema puede ser usado para adherirse a comunidades, si por ejemplo queremos camuflar nuestro Bot como una persona a la cual le gustan las bicicletas, podemos hacer que empiece a seguir cuentas que hablen bien sobre bicicletas, y mal sobre coches.

10.4 Sistema propuesto

Nos limitamos a crear un sistema que calcula una reacción. La acción consecuente depende de su implementación.

Proponemos un sistema que, dado un texto y los intereses a simular, genera una reacción. Antes de explicar el sistema debemos presentar como medimos los intereses, y las reacciones.

Tratamos una reacción como un vector de dos valores, la *Polaridad* y la *Intensidad*, ambos comprendidos en el intervalo (0-1). La polaridad representa si es una reacción buena/mala, y la intensidad su fuerza. Podemos ver una simplificación en la siguiente figura.

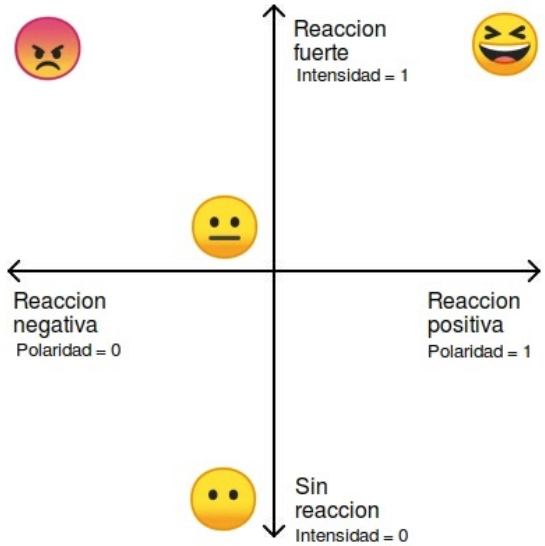

Ilustración 11: Ejes de los valores de una reacción

Un Interés también tiene valores de *Polaridad* e *Intensidad*, y relaciona estos con palabras clave que simbolizan ese interés.

Siguiendo con el ejemplo anterior, si tenemos una cuenta a la cual le gustan las bicicletas y detesta los coches, sus dos intereses pueden ser:

Palabras Clave	**Intensidad**	**Polaridad**
Bicicleta, bici, tándem	0.7	0.8
Coche, vehículo, automóvil	0.9	0.01

Tabla 5: Ejemplo de intereses

10.5 Implementación del sistema

El código estará implementado en nuestro paquete ***natural_Bot_tk*** dentro del submódulo ***reaction*.**

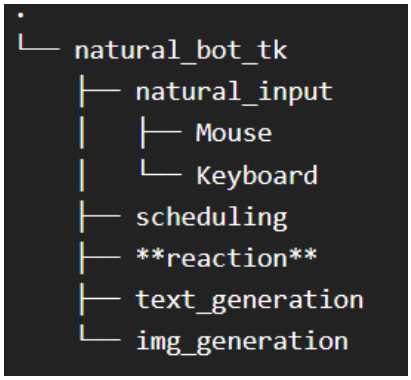

Ilustración 12: Localización del sistema de reacción.

Para tratar con intereses, hemos creado una clase llamada **Interest**.

Una vez tenemos los intereses de nuestro Bot, el proceso que seguimos para estimar la reacción a un contenido cualquiera es el siguiente:

1. Extraemos los temas principales del contenido.
2. Calculamos cual es el sentimiento (positivo/negativo) del contenido
3. Si hay, tomamos el tema más ocurrente que está en nuestros intereses.
4. Calculamos cuánto se ajusta el sentimiento del contenido a nuestro interés del tema, y con ello calculamos una reacción.

Este sistema se puede aplicar a todo tipo de contenido, solo hay que conseguir como extraer los temas de los que trata, y su sentimiento. Con imágenes esto es especialmente difícil.

Nos hemos centrado en adaptar este sistema a texto, a continuación, explicamos cómo funciona esta adaptación:

1. Extraemos los temas principales extrayendo los sustantivos con **NLTK**, una librería para lenguaje natural, y dejando solo sus raíces. Posteriormente los ordenamos según el número de ocurrencias.
2. Estimamos el sentimiento del texto mediante un clasificador de Bayes incluido en **TextBlob**, otra librería para lenguaje natural.
3. Si hay, tomamos el tema con más ocurrencias que está en nuestros intereses.
4. Calculamos cuánto se ajusta el texto a nuestra percepción del tema, y calculamos una reacción.

Para calcular la reacción hemos desarrollado un sistema con los siguientes pasos:

1. Calculamos como de extremista (R) es nuestro interés.

$$R(polarity) = |p - 0.5| \,/0.5$$

Ecuación 3: Formula que calcula el extremismo

2. Calculamos como de correcto (L) nos parece el texto según nuestro interés. Esto simula que vemos como correcto lo que comparte nuestras opiniones

$$L(P_I,\ P_t) = \frac{1 + sign(P_I - 0.5)}{2} - P_t$$

$$P_I := Polaridad\ del\ Interes$$
$$P_t := Polaridad\ del\ texto$$

Ecuación 4: Formula que calcula la percepción

3. Calculamos la polaridad de nuestra reacción.

$$P_r(P_I, P_t) = R(P_I) * L(P_I, P_t) + 0.5 * (1 - R(P_I))$$

$$P_I := Polaridad\ del\ interes$$
$$P_t := Polaridad\ del\ texto$$

Ecuación 5: Formula que calcula la polaridad de la reacción

4. Por último, calculamos la intensidad de nuestra reacción.

$$S_r(P_r, S_I) = 0.5 * R(P_r) + 0.5 * S_I$$

$$P_r := Polaridad\ de\ la\ reaccion$$
$$S_I := Intensidad\ de\ nuestro\ interes$$

Ecuación 6: Formula que calcula la intensidad de la reacción

El prototipo de la función que implementa este procedimiento es la siguiente:

```
def estimate_reaction(text, interest_list, bayes=True,
debug=False):
    ''' Estima la reacción a un texto dados los intereses de la
persona. \n
    `text`: String con el texto\n
    `bayes`: Calcular el sentimiento con clasificador bayesiano\n
    `interest_list`: Lista de Interests
    '''
```

El sistema está programado para textos en inglés, ya que es el idioma más común en el internet.

10.6 Ejemplos

Imaginemos que queremos hacer un Bot que reacciona como el sobrino de 5 años del autor; sin simplificar, sus intereses son los siguientes:

Palabras clave	**Intensidad**	**Polaridad**
firefighter	0.9	0.9
fire	0.9	0.1
car	0.6	0.8
trucks	0.6	0.7
dogs	0.9	0
ducks	0.9	0.1

El sistema automáticamente añade sinónimos a las palabras clave.

A continuación, mostramos el resultado de pasar cada texto por el sistema:

estimate_reaction(text, interests, bayes=False)

Texto	**Intensidad de la reacción**	**Polaridad de la reacción**
I'm so excited to have a dog, he is so cute!!	0.7390625	0.2109375
The fireman came to save me, they are so strong.	0.6233333	0.67333333
Ducks are the worst and we shouldn't give them bread.	0.85	0.9
Let's go for a ride in a racing car.	0.45	0.5
My house was on fire and now im sad	0.65	0.7
The stock market reached an all time high	0.0	0.0

11. Generación de contenido

11.1 Introducción

La última medida de detección que vamos a tratar es posiblemente la más difícil de engañar, los humanos.

Al ser RRSS, la mayoría de las cuentas con las que los Bot interactúan son controladas por personas. Las personan pueden denunciar una cuenta como sospechosa si consideran que su comportamiento o contenido es errático, lo que supondría que la cuenta fuera estudiada con detenimiento por un moderador, cosa que no deseamos.
Los humanos no se fijan en cómo movemos el ratón, ni tampoco en nuestra huella digital; se fijan en el contenido que subimos a nuestra cuenta.

Ejemplos de contenido que pueden delatar a un Bot es que sus fotos están sacadas de internet sin contexto, o que publique texto monotemático que se repite.

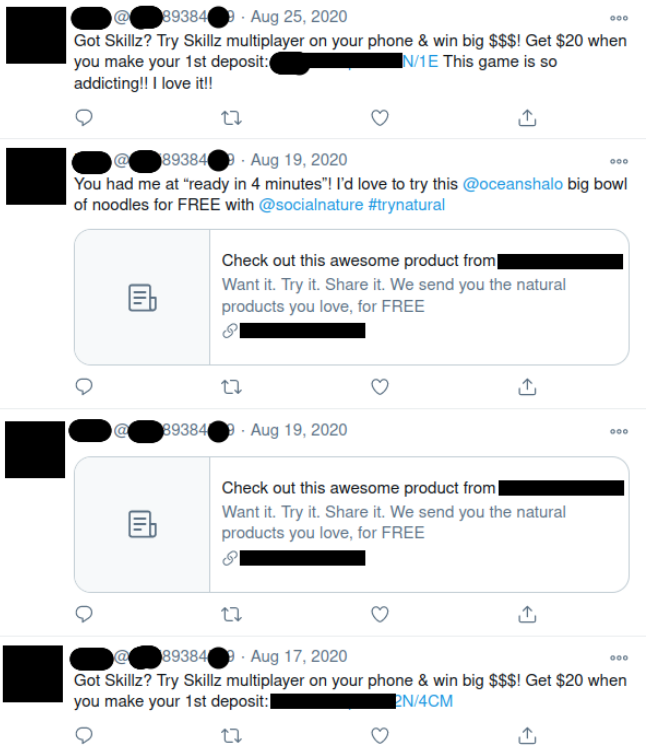

Ilustración 13: Ejemplo de contenido publicado por un Bot

Para terminar con las funcionalidades de nuestra librería, implementaremos dos sistemas por los cuales seamos capaces de generar contenido. Este no tiene como objetivo engañar en una inspección de cerca, pero si aparentar ser contenido lo suficientemente realista como para no crear sospechas a un usuario que observa el perfil rápidamente.
Podemos imaginar el objetivo de estas medidas como el de la tela de camuflaje, no es un pino, pero en un bosque pasa desapercibido.

11.2 Generación de texto

Nuestro objetivo es hacer una herramienta que genera texto de apariencia coherente dados unos temas. Para entender la solución primero debemos entender el desafío que presenta el lenguaje natural.

11.2.1 Lenguaje natural

Consideramos lenguaje natural a aquel lenguaje generado espontáneamente por los humanos con el propósito de comunicarse. Al haber sido creado sobre la marcha, pueden presentar ambigüedad y una complejidad difícil de igualar con lenguajes formales como la lógica formal.
Esta complejidad y ambigüedad suponen que procesarlos con ordenadores es extremadamente complicado.
La forma más simple de procesar lenguaje natural es convirtiendo las palabras en tokens y usando procesos estadísticos. Un ejemplo de estos métodos son las librerías **NLTK** y **TextBlob** usadas en el **Capítulo 10** de este trabajo para generar reacciones a texto.

Actualmente la forma más poderosa para generar lenguaje natural son los modelos GPT de OpenAI [21].

11.2.2 Introducción a GPT-2

Generative Pre-trained Transformer o GPT es, como el nombre indica, un transformador. Los transformadores son un modelo de red neuronal que funciona pasando la entrada por múltiples capas que la codifican a un valor intermedio, y posteriormente pasando ese valor intermedio por múltiples capas que lo decodifican al resultado.

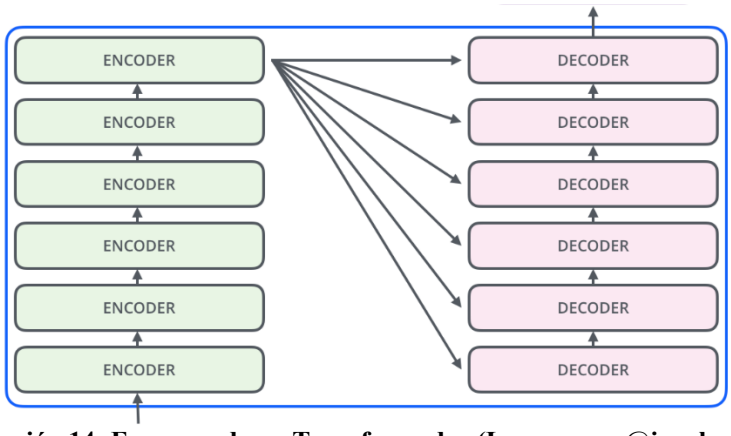

Ilustración 14: Esquema de un Transformador (Imagen por @jayalammar)

Los modelos de GTP se pueden usar para muchos fines, como el procesamiento de texto, generar imágenes a partir de un texto [22], o en nuestro caso, generar texto a partir de unos temas.

GTP tiene varias versiones, la última y más complicada, GPT-3, aun no es pública. Nosotros usaremos GPT-2, y dentro de ella, el modelo de 117 millones de parámetros, el más pequeño.

11.2.3 Solución propuesta

Proponemos un sistema por el cual generar texto natural dados unos temas usando GPT-2. El texto de la entrada y salida será inglés.

Para crear este sistema hemos usado como base el modelo pequeño ya entrenado de GPT-2.
Mientras que este modelo está entrenado, lo único que hace es continuar el texto que le damos como entrada.

Nuestro objetivo es que le podamos dar como entrada unos temas separados por comas y que su salida sea un texto corto que presente esos temas. Por ejemplo:

Entrada	**Salida**
christmas, snow, great, love	I love snow in christmas

Tabla 6: Ejemplo de Entrada/Salida deseada

Para ello tenemos que reentrenar el modelo sobre ejemplos de texto con esta estructura.
Para ello, necesitamos los suficientes textos cortos para reentrenar GPT-2.
Para solucionarlo, hemos usado una colección de Tweets creada por Sentiment140 [23], a los cuales hemos extraído los sustantivos y verbos, dejando cada tweet de la forma:

```
Input: body, feels, itchy, like, fire
Output: my whole body feels itchy and like its on fire
<|endtext|>

Input: spring, break, city, snowing
Output: spring break in plain city... it's snowing
<|endtext|>

Input: whole, crew, @mention
Output: @mention not the whole crew
<|endtext|>

Input: need, hug
Output: Need a hug
<|endtext|>
```

Tabla 7: Primeras entradas del set de datos usado

Con 30.000 tweets preparados de esta forma, hemos usado el código de ***nshepperd*** [24] que nos facilita reentrenar el modelo pequeño.

El tiempo de entrenamiento fue de 15 horas sobre un portátil; lo que supusieron 1200 epoch (iteraciones), y un error de 1.7. El error define como de preciso es nuestro modelo, cuanto menos, mejor.

Un error de 1.7 no es bueno, pero es suficiente para que el modelo entienda nuestra estructura, y para estudiar la viabilidad de este sistema para una futura versión.

11.2.4 Implementación

El código estará implementado en nuestro paquete ***natural_Bot_tk*** dentro del submódulo ***text_generation.*** Y el código necesario para gpt2 en ***resources/gpt2***.

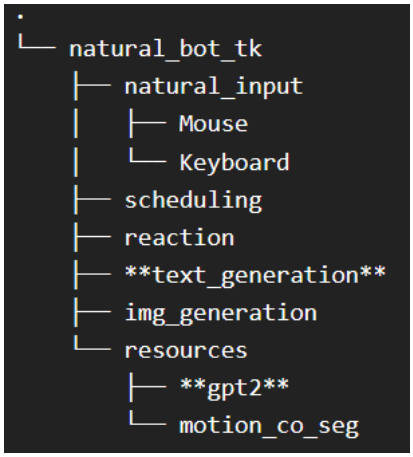

Ilustración 15: Localización del código de síntesis de texto

El primer paso fue modificar el código fuente de GPT-2 para que funcione con las últimas versiones de Tensor Flow.

Una vez tenemos un modelo entrenado, solo queda probar si funciona. Hemos simplificado la interacción con el modelo creando una clase llamada GPT2. A continuación, mostramos el prototipo de la función de esa clase usada para introducir input al modelo.

```
def prompt_model(self, prompt,
          temperature=1,
          top_k=0,
          top_p=0.0):
      ''' Con el modelo de GPT dado generar una salida.\n
      `prompt`: Texto a usar como input\n
      `temperature`=1 : Valor controlando la aleatoriedad de la
                        salida\n
      `top_k`=0 : Integer que controla la diversidad. Representa
                  el número de palabras consideradas. 0 = sin
                  restricción\n
      `top_p`=0.: Float que controla la diversidad. Un buen valor
                  es 0.9.'''
```

11.2.5 Ejemplos

A continuación, exponemos algunos ejemplos de entrada-salida de este sistema. Damos el valor 1 a ***temperature***, por lo que la salida no es determinista.

Entrada	**Salida**
snow, city, christmas, great	poor Ski Staff. It's snowing outside in downtown christmas city again (great Mavericks weather) - great hours.
firetruck, fireman, house, cat	The most active fireman on House Alecares yesterday. Less than three houses asleep. No catout here. House Catoo!
green, bike, stolen, help	@mention YOU! i bike for them! your bike rident stolen them so i like it! IM HELP!
green, bike, stolen, help	@mention GPG encrypted wifi is in a broken bike. I stolen my bike from it HELP ME!!!
white, rabbit, carrot	@mention What about the rabbits and true boars? or the true orange ones? Still orange though!

La salida no suele ser muy coherente, pero demuestra que es viable entrenar un modelo para generar pequeñas frases dados unos temas. Mejorar este sistema es fácil, podemos usar un modelo de GPT-2 mayor, usar más tweets en el entrenamiento, o entrenar el modelo durante más tiempo supondrán un mejor resultado.
Mejorar este sistema no supone la generación de texto completamente coherente, pero si mejor.

11.3 Generación de imágenes

No todas las RRSS se basan en texto, otras como Pinterest o Instagram están basadas en imágenes, nuestro desafío es conseguir un sistema por el cual los Bots que lo usen puedan subir fotos relativamente coherentes en los que aparezcan ellos (o su avatar).

Subir imágenes aleatorias de internet puede funcionar, pero no es una solución estable si el Bot se está haciendo pasar por un humano.
Queremos desarrollar un sistema que le dé a los Bots la capacidad de subir fotos de "sí mismos" en diversas situaciones. Para ello el sistema tomará una imagen en la que salen personas ajenas, y sustituye una de las caras por la cara del avatar del Bot.

Suponemos que nuestro Bot tiene un avatar como los generados por StyleGAN2 [25]

Ilustración 16: Cara de persona generada por StyleGAN2 en thispersondoesnotexist.com

11.3.2 Introducción al intercambio de caras

En los últimos años una de las prestaciones más usadas en RRSS como Instagram y Snapchat son los filtros. Estos nos dan la posibilidad de añadir efectos sobre imágenes de nuestras caras, como ponernos gafas o bigote.

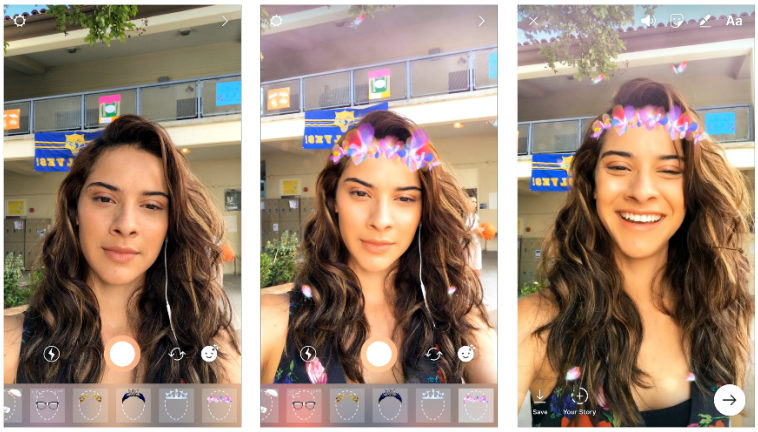

Ilustración 17: Filtros de Instagram (por about.instagram.com)

Estos funcionan detectando los puntos de referencia faciales. Estos se utilizan para localizar y representar las regiones salientes de la cara, como por ejemplo la mandíbula, los ojos y la boca.

A parte de usarlo para filtros, estos puntos de referencia pueden ser usados para crear una máscara de la cara, que posteriormente se puede aplicar sobre otra. Esta es la forma más simple de intercambiar dos caras.
Mientras que este método es ampliamente usado, sus resultados no son muy convincentes.

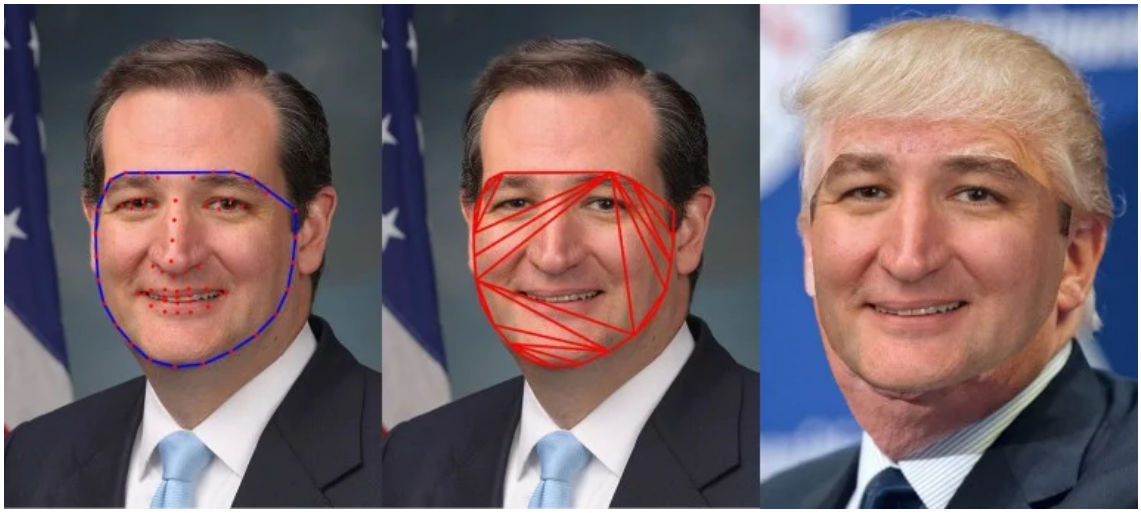

Ilustración 18: (a) Puntos de referencia faciales (b) Mascara de la cara (c) Intercambio de caras (por learnopencv.com)

11.3.3 Introducción al uso de Autoencoders

Otra opción para intercambiar dos caras es el uso de redes neuronales. Normalmente se usan Autoencoders.

Un Autoencoder es un algoritmo de aprendizaje no supervisado cuyo objetivo es aprender a hacer coincidir la salida con la entrada. Para ello convierten la entrada multidimensional a un valor intermedio de menores dimensiones; e intentan reconstruir la entrada a partir del valor de menos dimensiones.

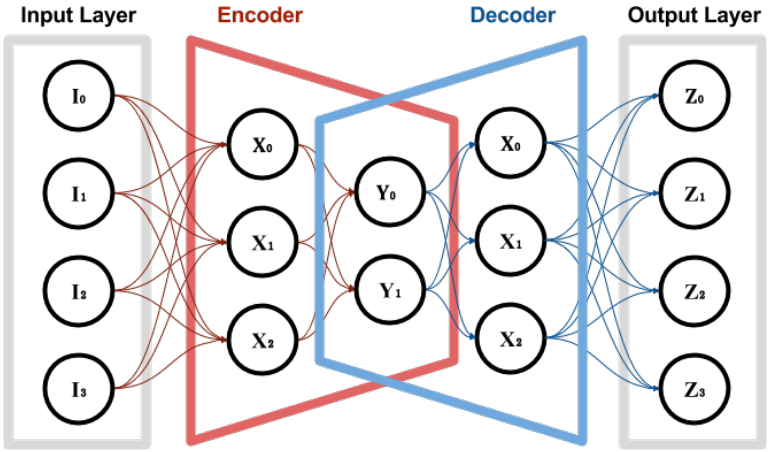

Ilustración 19: Simplificación de las capas de un Autoencoder (Imagen por Alan Zucconi)

Podemos usar este mismo procedimiento para codificar nuestras caras en un valor intermedio (latente).

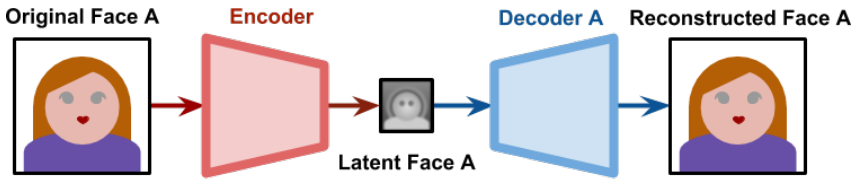

Ilustración 20: Uso de un Autoencoder en una cara (Imagen por Alan Zucconi)

Las capas decodificadoras del Autoencoder son capaces de generar una cara a partir de este valor intermedio (latente). Ahora bien, si sobre las capas decodificadoras de una cara A usamos el valor intermedio de una cara B, la red reconstruye la cara A con la posición y expresión de la cara B.

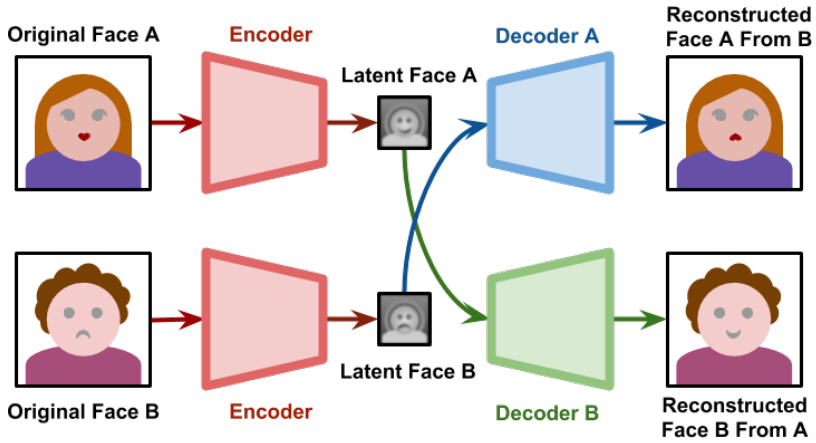

Ilustración 21: Uso de un Autoencoder para intercambiar caras (Imagen por Alan Zucconi)

Si queremos cambiar dos caras, solo nos queda editar una cara sobre la otra en un proceso que funciona igual que el que usaba puntos de referencia, generando una máscara de la cara generada y sustituyendo la original.

Mientras este método es mucho más realista que el anterior, queremos un procedimiento más flexible, y que sea capaz de cambiar más partes de la cabeza a parte de la cara.

11.3.4 Solución propuesta

Para nuestro propósito usaremos un modelo de segmentación no supervisado presente en el artículo ¨**Motion-supervised Co-Part Segmentation**¨ [26]
El objetivo de este modelo es ser capaz de segmentar imágenes en zonas sin necesidad de supervisión, ya que usa la información del movimiento inferida de videos para descubrir cuáles las zonas significativas del objeto en una imagen.

Este método tiene dos partes, el módulo de segmentación, cuyo objetivo es el de segmentar cada imagen del video en las zonas significativas; y el módulo de reconstrucción, cuyo objetivo es reconstruir la imagen a partir de estas zonas significativas.

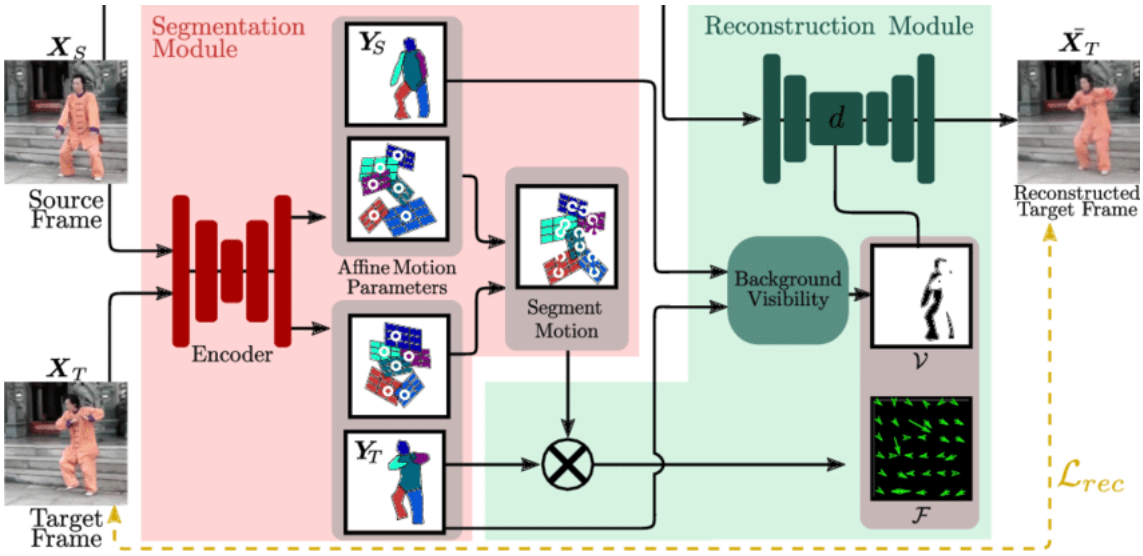

Ilustración 22: Esquema del modelo presentado en Motion-supervised Co-Part Segmentation

Como podemos ver, el modelo está construido con Autoencoders, y los dos módulos funcionan como uno.

Este método puede ser usado para segmentar caras en zonas significativas, y sustituir esas zonas entre caras. Para este cometido ellos facilitan 3 modelos ya entrenados, de los cuales usaremos el que segmenta caras en 10 zonas.

Ilustración 23: Cara a ser segmentada

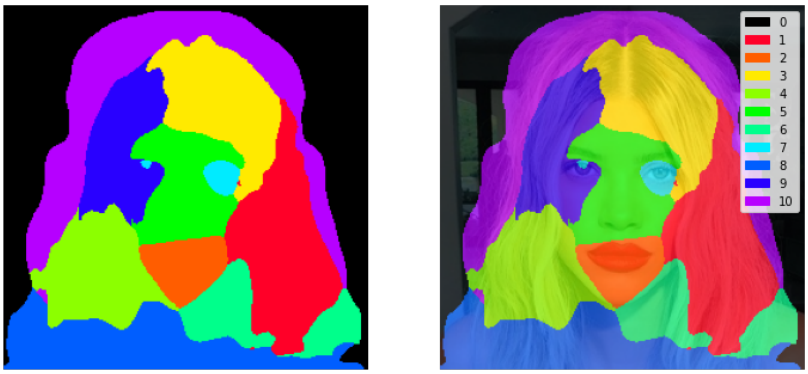

Ilustración 24: Cara segmentada en zonas significativas

11.3.5 Implementación

El código estará implementado en nuestro paquete ***natural_Bot_tk*** dentro del submódulo ***img_generation*.** Y el código necesario para el modelo en ***resources/motion_co_seg***.

```
.
└── natural_bot_tk
    ├── natural_input
    │   ├── Mouse
    │   └── Keyboard
    ├── scheduling
    ├── reaction
    ├── text_generation
    ├── **img_generation**
    └── resources
        ├── gpt2
        └── **motion_co_seg**
```

Ilustración 25: Localización del código de síntesis de imágenes

Proponemos aplicar este modelo como sistema para sustituir caras en dos imágenes, dada la imagen de un rostro como cara original, y una imagen de una o varias personas como objetivo. A continuación, mostramos las imágenes que usaremos en esta explicación.

Ilustración 26: Cara original, generada por StyleGAN2

Ilustración 27: Imagen objetivo

Los pasos que sigue nuestro sistema son los siguientes:

Paso 1: Detección de caras y selección de la más similar.
Para ello usamos la librería de **face_recognition** [27], que implementa funcionalidades que permiten detectar caras y calcular su similitud.

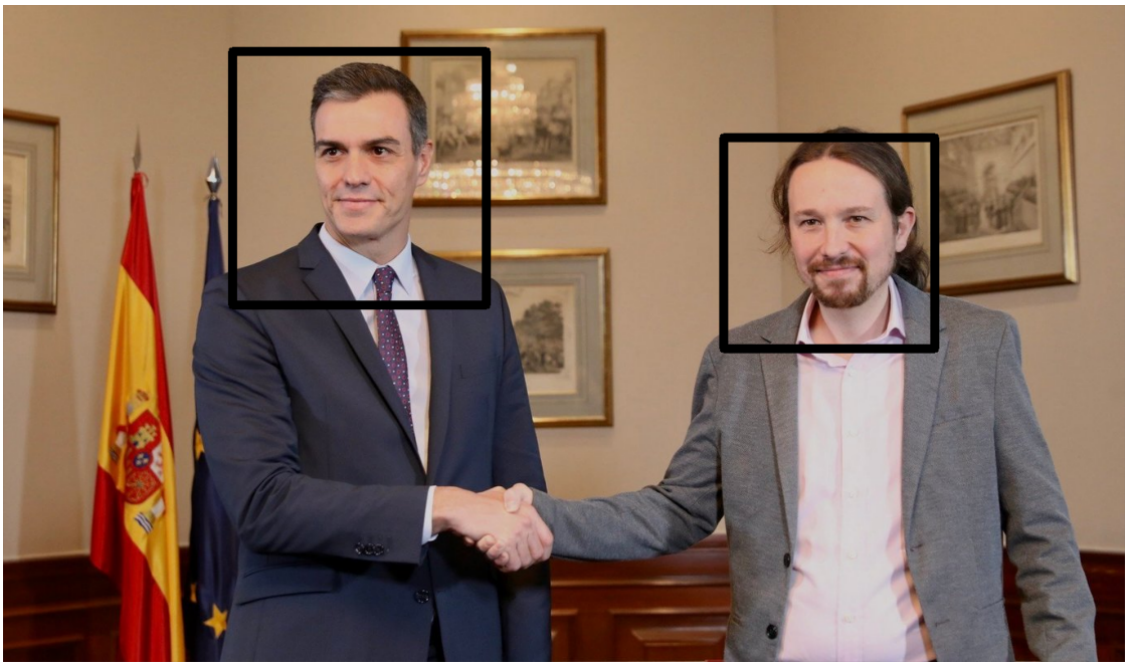

Ilustración 28: Detección de caras con face_recognition

El sistema automáticamente escoge la cara con mayor similitud. En este caso es la de la derecha.

Paso 2: Extracción y segmentación de ambas caras
Extraemos la cara de la derecha usando el recuadro como límites. Una vez tenemos ambas caras del mismo tamaño, usamos el modelo de segmentación [26] para separar ambas caras en zonas.

Ilustración 29: Imagen original segmentada

Ilustración 30: Imagen objetivo-segmentada

Paso 3: Sustitución de zonas
Por último, usamos el módulo de reconstrucción para sustituir las zonas de una imagen a la otra. Este módulo nos permite seleccionar qué zonas queremos sustituir, para nuestra implementación seleccionamos aquellas que corresponden a la cabeza, dejando el cuerpo y el fondo intacto.

Ilustración 31: Resultado de sustituir las caras

Podemos ver como se copian rasgos como el estilo de pelo, orejas, y se ajusta perfectamente a cara objetivo. También podemos ver desperfectos como la mirada perdida y aun ligeros tonos de la barba.

Paso 4: Reconstrucción de la imagen
Por último, reconstruimos la imagen posicionando esta cara editada sobre su posición original.

Ilustración 32: Imagen final
Se puede observar el recuadro de la imagen sustituida, esto será solucionado en futuras versiones.

El código dentro de ***img_generation*** contendrá tres funciones:

```
def replace_best_face(mugshot_path, target_img_path, threshold=0.9,
gpu=False):
   ''' Remplaza una cara sobre la más similar en una imagen
objetivo.\n
   `mugshot_path`: Ruta de la foto de la cara.\n
   `target_img_path`: Ruta de la foto objetivo.\n
   `threshold`: Similaridad mínima para sustituir caras\n
   `gpu`: Usar gpu
   '''
```

```
def replace_face(mugshot_path, target_img_path, face_location,
gpu=False):
   ''' Remplaza una cara sobre otra en una imagen dada la
localización de
   la cara objetivo.\n
   `mugshot_path`: Ruta de la foto de la cara.\n
   `target_img_path`: Ruta de la foto objetivo.\n
   `face_location`: Array con la localización de la cara objetivo\n
   `gpu`: Usar gpu
   '''
```

```
def find_similar_faces(mugshot_path, target_img_path):
   ''' Encuentra caras en la imagen objetivo, y compara su
similaridad con
   la foto de la cara.\n
   `mugshot_path`: Ruta de la foto de la cara.\n
   `target_img_path`: Ruta de la foto objetivo
   '''
```

11.3.6 Ejemplos

A continuación, mostramos algunos ejemplos del uso de este sistema.

11.6.3.1 Ejemplo 1

Ilustración 33: Imagen original 1, generada por StyleGAN2

Ilustración 34: Imagen objetivo 1 (por Steven Van en Unsplash)

Ilustración 35: Imagen resultante 1

11.6.3.2 Ejemplo 2

Ilustración 36: Imagen original 2, generada por StyleGAN2

Ilustración 37: Imagen objetivo 2 (por Roland Samuel en Unsplash)

Ilustración 38: Imagen resultante 2

11.3.6.3 Ejemplo 3

Ilustración 39: Imagen original 3, generada por StyleGAN2

Ilustración 40: Imagen objetivo 3 (por Brooke Cagle en Unsplash)

Ilustración 41: Imagen resultante 3

12. Estructura del paquete

12.1 Introducción

Hemos decidido programar esta librería de funciones en Python, debido a su simplicidad, y su amplio uso para programar Bots.

Python, al ser un lenguaje interpretado, permite importar funciones y valores de cualquier archivo ***.py***, al los cuales denominamos modulos.
A continuación, podemos ver como se importaría un modulo en Python.

```
import module
```

Python soporta espacios de nombres. Un espacio de nombres es un prefijo usado para agrupar módulos con un mismo prefijo. Para crear este prefijo tenemos que guardar los módulos dentro de una misma carpeta, en la que haya un archivo llamado ***__init__.py***. Python reconocerá esta carpeta como un paquete, y el nombre de la carpeta será el nombre del espacio de nombres.

```
from package import module
```

12.2 Organización de nuestros módulos

Agrupamos nuestro código en diferentes módulos según su función:

natural_input	Agrupa las clases necesarias para imitar el input humano: Keyboard y Mouse.
reaction	Contiene el código necesario para calcular reacciones a partir de intereses.
scheduling	Contiene el código necesario para generar activaciones.
img_generation	Agrupa el código necesario para usar los modelos que usamos para generar imágenes.
text_generation	Contiene una clase GPT2 que usamos para interactuar con el modelo GPT-2.

Estos módulos irán dentro de una carpeta con el nombre de nuestro paquete.
Además de estos, necesitamos una carpeta donde guardar el código auxiliar que necesitamos, pero que no queremos que sea fácilmente accesible, como aquel para interactuar con los modelos. Para ello agrupamos este código dentro de un sub-paquete llamado resources.

La estructura final es la siguiente:

```
.
└── natural_Bot_tk/
    ├── natural_Bot_tk/
    │   ├── __init__.py
    │   ├── natural_input.py
    │   ├── reaction.py
    │   ├── scheduling.py
    │   ├── text_generation.py
    │   ├── img_generation.py
    │   └── resources/
    │       ├── __init__.py
    │       ├── keyboard_layouts.py
    │       ├── motion_co_seg/
    │       │   └── [...]
    │       └── gpt2/
    │           └── [...]
    ├── notebooks/
    │   └── [...]
    ├── setup.py
    ├── requirements.txt
    ├── LICENSE.txt
    └── README.md
```

12.3 Instalación

Python busca los paquetes a importar en la carpeta donde se ejecuta, y en una carpeta global de su interprete. Para poder acceder a los modulos de nuestro paquete, debemos instalarlo dentro de esa carpeta global.

PIP, es la herramienta que gestiona paquetes en Python. Para poder instalar nuestro paquete debemos crear un archivo llamado ***setup.py***, en la raíz del paquete, que define sus metadatos.

A continuación, podemos ver el breve código de ***setup.py***

```
from setuptools import setup, find_packages

with open("requirements.txt", 'r') as req_f:
    requirements = []
    for l in req_f:
        requirements.append(l.strip())

setup(
    name='natural_Bot_tk',
    version='0.1',
    packages=find_packages(exclude=['notebooks']),
    license='MIT',
    description='Natural Bot behaviour toolkit',
    long_description=open('README.md').read(),
    install_requires=requirements,
```

```
    url='https://github.com/RicardoStefanescu/natural_Bot_tk',
    author='Ricardo Stefanescu',
    author_email='ricardo.stefanescu@edu.uah.es'
)
```

Como podemos ver, nuestro paquete tiene una serie de paquetes requeridos para poder ser instalado. Por facilidad, guardamos sus nombres dentro de un archivo llamado ***requirements.txt***.
Estos paquetes requeridos incluyen todos aquellos que usamos en nuestro código por una función o otra, como tensorflow para cargar y usar los modelos de aprendizaje maquina.

12.4 Licencia

Para terminar, debemos añadir una licencia de uso a nuestro software. Como es open-source, y deseamos ver que hace la comunidad con el, usamos la licencia **GPLv3**. Esta licencia supone que solo se podrá usar para proyectos open-source.

13. Conclusión

Nunca podremos comprobar que esta librería será usada con intenciones benignas. Los Bots en las redes sociales suponen un problema en todos los aspectos humanos, y como podemos ver, cada vez existe más tecnología que puede ser usada para camuflarlos como humanos.

Las medidas de detección de Bots funcionan hasta un punto, y no podemos prohibir esta tecnología con el objetivo de deshacernos de los Bots. Posiblemente la mejor solución contra ellos es educar a la población sobre su existencia, y sus consecuencias.

Hemos desarrollado una librería con el fin de contrarrestar ligeramente las medidas de detección actuales, aun así, existe una carrera entre las medidas de detección y las contramedidas, por lo que esperamos que esta librería sea insuficiente en un plazo corto de tiempo si no es actualizada constantemente.

14. Bibliografía

[1] L. Finger, «Do Evil - The Business Of Social Media Bots,» *Forbes,* 17 February 2015.

[2] O. V. C. D. F. M. A. F. Emilio Ferrara, «The Rise of Social Bots,» 2015.

[3] R. Bååth, «castle.io,» 2020. [En línea]. Available: https://blog.castle.io/Bot-or-not-can-you-spot-the-automated-mouse-movements/.

[4] A. M. J. F. R. V.-R. Alejandro Acien, «BeCAPTCHA-Mouse: Synthetic Mouse Trajectoriesand Improved Bot Detection».

[5] E. E.-Z. Margit Antal, «Intrusion Detection Using Mouse Dynamics,» *IET Research Journals,* 2018.

[6] Z. C. X. G. Y. D. R. A. M. Chao Shen, «User Authentication Through Mouse Dynamics,» *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY,* 2013.

[7] J. Vali, «GitHub - Natural Mouse Motion,» [En línea]. Available: https://github.com/JoonasVali/NaturalMouseMotion/.

[8] V. Bavitz, «GitHub - BezMouse,» [En línea]. Available: https://github.com/xvol/bezmouse.

[9] A. Nazar, «Synthesis & Simulation of Mouse Dynamics,» 2003 .

[10] neuronio, «Generating mouse movement with ANN,» [En línea]. Available: https://medium.com/neuronio/generating-mouse-movement-with-ann-6659c53a6fe2.

[11] B. S. M.S Obaidat, «Verification of computer users using keystroke dynam-ics. IEEE Transactions on Systems, Man and Cybernetics,» *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* 1997.

[12] D. W. X. T. Dawn Xiaodong Song, «Timing Analysis of Keystrokes and Timing Attacks on SSH,» 2002.

[13] D. L. W. Salil P. Banerjee, «Biometric Authentication and Identification using Keystroke Dynamics: A Survey,» *Journal of Pattern Recognition Research 7 (,* 2012.

[14] M. E.-A. a. C. R. Romain Giot, «Keystroke Dynamics Authentication,» de *Biometrics*, D. J. Yang, Ed., 2011.

[15] Bablosoft, «Browser Automation Studio,» [En línea]. Available: https://bablosoft.com/shop/BrowserAutomationStudio.

[16] A. Sweigart, «PyAutoGui Documentation,» [En línea]. Available: https://pyautogui.readthedocs.io/en/latest/.

[17] A. M. T. P. K. T. Elie Bursztein, «Picasso: Lightweight Device Class Fingerprinting for Web Clients,» *Workshop on Security and Privacy in Smartphones and Mobile Devices,* 2016.

[18] Google, «Google ReCaptcha V3,» [En línea]. Available: https://developers.google.com/recaptcha/docs/v3.

[19] Radware, «Radware Bot Manager,» [En línea]. Available: https://www.radwarebotmanager.com/Bot-intelligence-feed/.

[20] N. B. B. B. G. A. Pierre Laperdrix, «Browser Fingerprinting: A survey,» 2019.

[21] J. W. R. C. D. L. D. A. I. S. Alec Radford, «Language Models are Unsupervised Multitask Learners,» 2018.

[22] M. P. G. G. S. G. Aditya Ramesh, «OpenAI DALL-E,» 2021. [En línea]. Available: https://openai.com/blog/dall-e/.

[23] Sentiment140, «Sentiment140 Dataset,» 2017. [En línea]. Available: https://www.kaggle.com/kazanova/sentiment140.

[24] nshepperd, «GPT-2 Finetuning,» [En línea]. Available: https://github.com/nshepperd/gpt-2.

[25] S. L. M. A. J. H. J. L. T. A. Tero Karras, «Analyzing and Improving the Image Quality of StyleGAN,» 2019.

[26] S. R. S. L. S. T. E. R. N. S. Aliaksandr Siarohin, «Motion-supervised Co-Part Segmentation,» 2020 .

[27] A. Geitgey, «PyPi - face_recognition,» [En línea]. Available: https://pypi.org/project/face-recognition/.

[30] M. A. S. N. L. R. B. Abbas Abou Daya, «A Graph-Based Machine Learning Approach for Bot Detection,» 2019.